Wicked Fast PaaS

Performance Tuning of OpenShift v3 and Docker

Red Hat Performance Engineering

Fork me on GitHub

Environment Setup

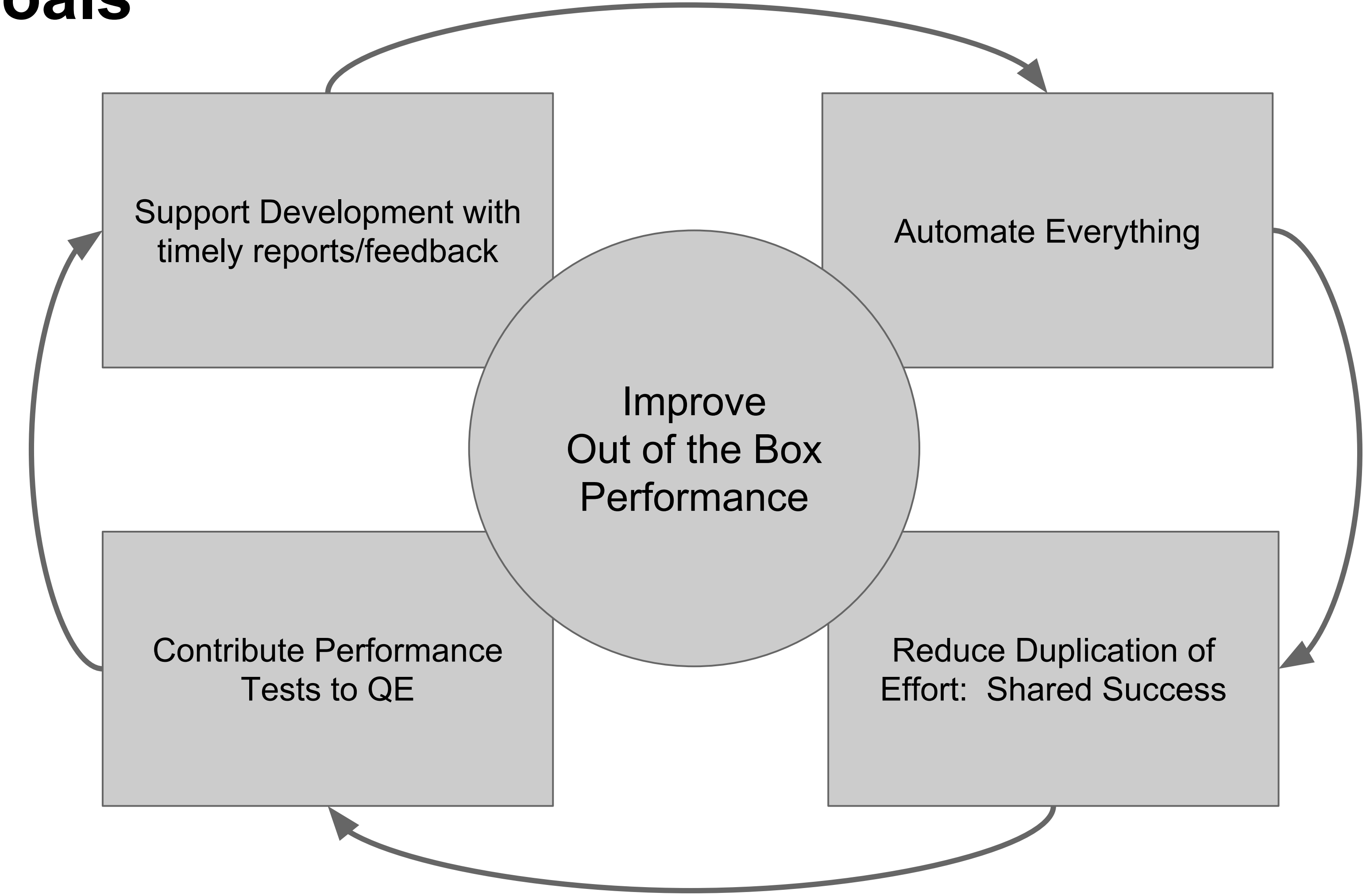
https://github.com/jeremyeder/openshift-performance

- Download OVA file (or copy from USB disk)
- Install VirtualBox and kernel-devel RPM that matches your running kernel
- systemctl restart system-modules-load or Reboot to load kernel modules
- Start VirtualBox
- Go to File -> Import Appliance -> Select the OVA file
- Click the checkbox to reset the MAC address
- Click Import and Start the VM
- Username: devconf2016 Password: devconf2016
- User devconf2016 has sudo access

Agenda

- Approach to Performance Analysis
- Latest Features
- Infrastructure Optimization
  - Compute, Network, Storage
- Tuning Docker and OpenShift
- Scaling OpenShift
- Architecture Overview

Goals



Support Development with timely reports/feedback

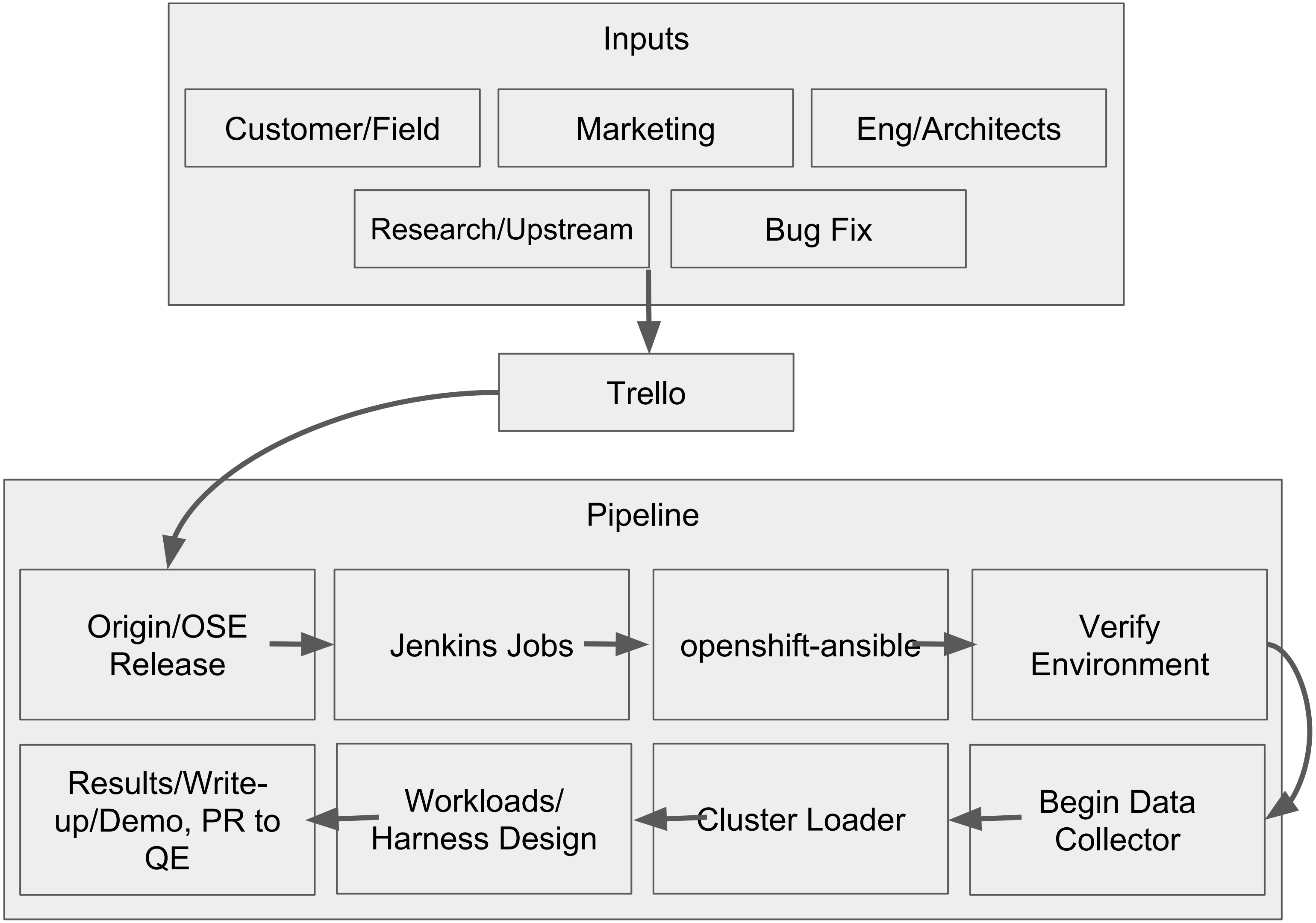
Automate Everything

Improve Out of the Box Performance

Reduce Duplication of Effort: Shared Success

Contribute Performance Tests to QE

Workflow



Inputs

Customer/Field

Marketing

Eng/Architects

Research/Upstream

Bug Fix

Trello

Pipeline

Origin/OSE Release

Jenkins Jobs

openshift-ansible

Verify Environment

Begin Data Collector

Cluster Loader

Workloads/ Harness Design

Results/Write-up/Demo, PR to QE

Tuning/Scaling fundamentals

<u>don’t change</u>

just for containers

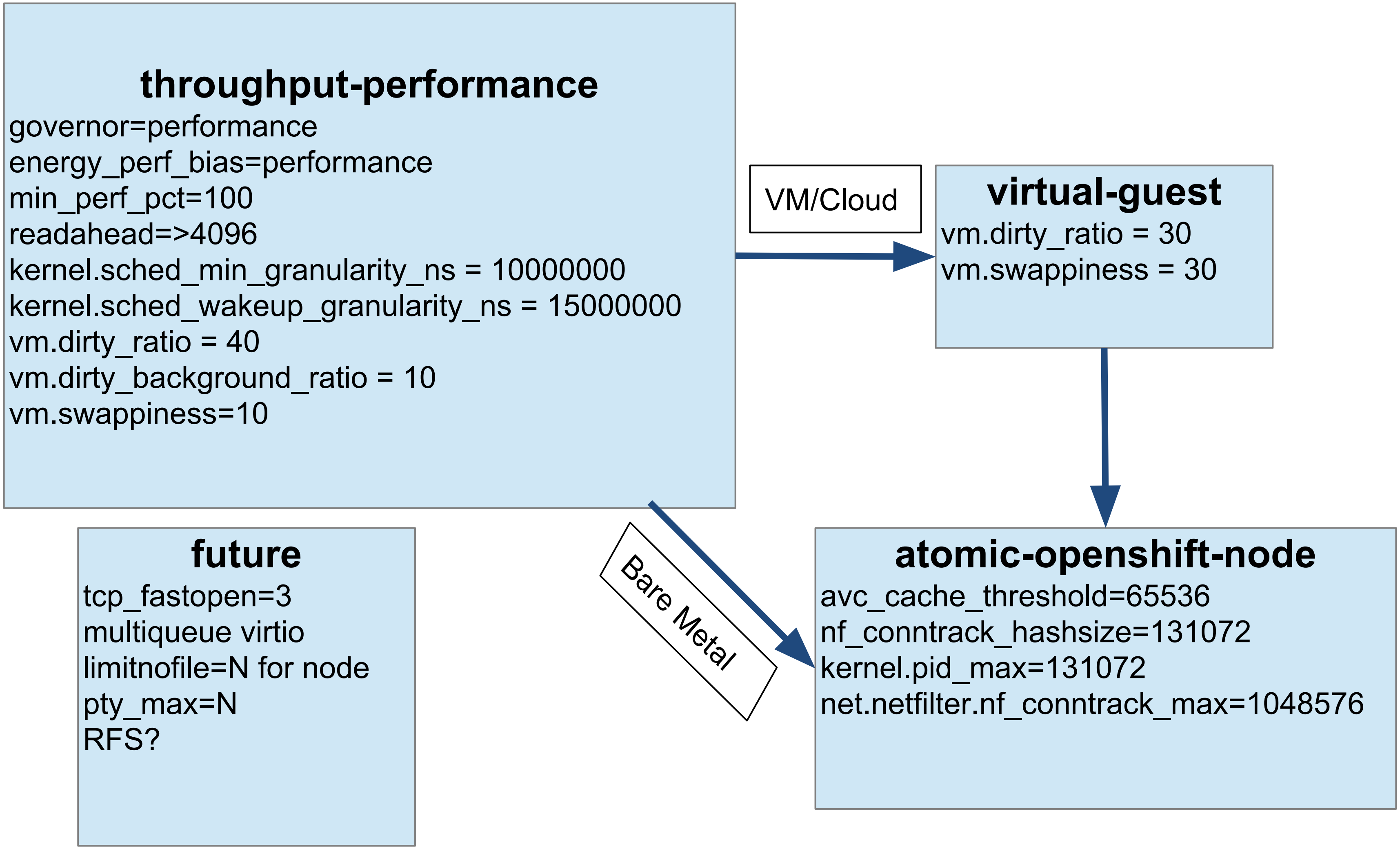
First: Tuning the Installer :-)

- Parallelizing is good and bad
  - Creates high load on content source.
- Installer node should be RHEL6.6 or later (ControlPersist)
- Pre-seed everything possible into your “gold image”
  - OS Updates, docker, docker-storage-setup, docker images, pre-register with Satellite/Content Source
- Ensure fast-access to content: Red Hat CDN/Satellite
- Ansible
  - Set forks ≥ nodes
  - Installer should be run on same LAN as cluster.
  - Increase ControlPersist to maintain persistent SSH connection

Ansible Config for Large Clusters

```
[defaults]
forks = 1000
host_key_checking = False
remote_user = root
roles_path = roles/
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /tmp/$USER_ansible/facts
fact_caching_timeout = 600
log_path = /tmp/$USER_ansible.log
[privilege_escalation]
become = False
[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=600s
control_path = %(directory)s/%%h-%%r
pipelining = True
```

Tuned Profiles for OpenShift



throughput-performance

governor=performance
energy_perf_bias=performance
min_perf_pct=100
readahead=>4096
kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 15000000
vm.dirty_ratio = 40
vm.dirty_background_ratio = 10
vm.swappiness=10

VM/Cloud

virtual-guest

vm.dirty_ratio = 30
vm.swappiness = 30

Bare Metal

atomic-openshift-node

avc_cache_threshold=65536
nf_conntrack_hashsize=131072
kernel.pid_max=131072
net.netfilter.nf_conntrack_max=1048576

future

tcp_fastopen=3
multiqueue virtio
limitnofile=N for node
pty_max=N
RFS?

Pbench

A Framework for Benchmarking and Performance Analysis

https://github.com/distributed-system-analysis/pbench

What is Pbench?

- pbench (perf bench) aims to:
  - provide easy access to benchmarking & performance tools on Linux systems
  - standardize the collection of telemetry and configuration information
  - automate benchmark execution
  - output effective visualization for analysis
  - allow for ingestion into elastic search

rhel-tools container

rhel-tools (and fedora-tools and centos-tools) are purpose-built analysis and debugging “super privileged containers”.

- strace, tcpdump, sysstat, sosreport, git

Overview and Official Documentation

tl;dr

```
# docker pull centos/tools
# atomic run centos/tools
```

Resource Management

```
$ cat openshift-performance/svt/content/quota-default.json
      "memory": "1Gi", # every pod can use 1Gi of memory
     "cpu": "20",  # “milli-cores”
     "pods": "10", # max pods
     "services": "5",  # max services
     "replicationcontrollers":"5", # max rc’s
     "resourcequotas":"1" # max quota objects
```

https://github.com/kubernetes/kubernetes/blob/master/docs/design/resources.md

CPU/Memory Optimization

- RHEL7 task scheduler adds automatic numa_balancing
- Pod commands can use numactl
    - docker has support for --cpuset-cpus and --cpuset-mems
    - Not in Kube yet
    - Pod manifests can use nodeSelector w/node labels to land on fast gear

Storage Optimization

- *Ensure you are using thinLVM (not loopLVM)*
- Persistent data gets stored in “Persistent Volumes”
  - Ceph/Gluster/NFS/iSCSI/Fiber
  - Bind-mounted into container at startup

- Container storage I/O plays by the same rules as always
- I/O scheduler and others (vm.dirty etc) are system-wide
- If a container has a very particular tuning need, consider dedicated resources (HostPath pass-through)

Docker Graph Driver

- Pluggable image/container storage backend

- Device Mapper
  - Use docker-storage-setup, which will setup “thinLVM”
  - Supported in RHEL7.0+, SELinux and POSIX compliant
- Overlay FS
  - Supported (with important caveats) as of RHEL7.2
  - Increased density, faster container start/stop (page cache sharing)
  - Non-POSIX compliant, no SELinux support
- Comparison https://developerblog.redhat.com/2014/09/30/overview-storage-scalability-docker/

Network Optimization

- OpenShift and Atomic Enterprise “just need connectivity”
- We use OpenvSwitch w/VXLAN tunnels
- VXLAN handles 1G pipes w/o issue
  - 10G+ needs tuning: VXLAN-offload, faster CPUs, jumbo frames
- Container network I/O plays by the same rules as always
  - NIC-level tuning such as jumbo frames/offloads are interface-wide
  - Kernel sysctl tunings are often per-container (somaxconn), sometimes not (tcp_mem)
  - You can share host network stack or use kernel-bypass into a container (Solarflare OpenOnload/Intel DPDK)
  - http://developerblog.redhat.com/2015/04/09/accelerating-rhel7-linux-containers-solarflare-openonload/
  - http://developerblog.redhat.com/2015/06/02/can-you-run-intels-data-

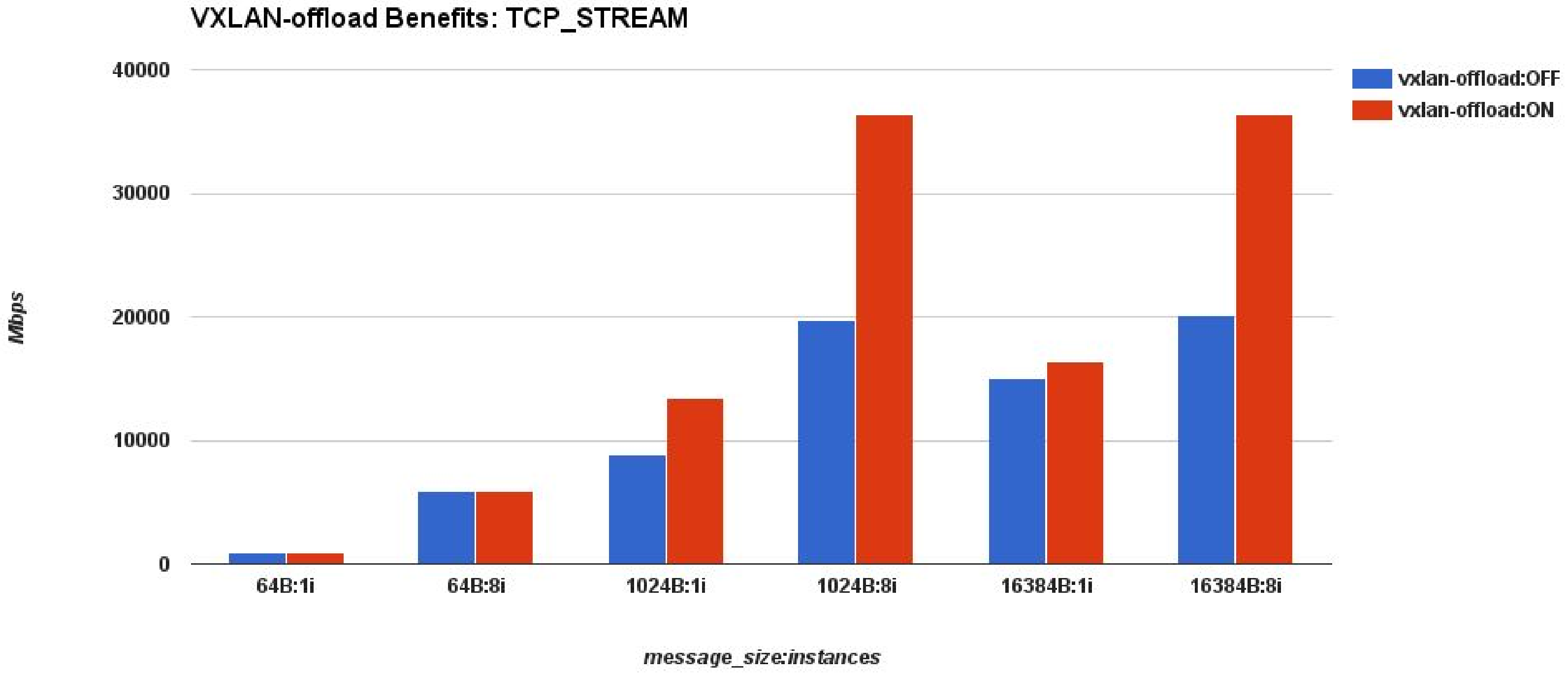
VXLAN-offload

- Certain NICs have VXLAN-offload capabilities in hardware
  - High-end models from Intel, Mellanox, Emulex, and RHEL7.1+

- VXLAN-offload handles packet checksums on the NIC rather than the CPU
- Another in a long line of hardware-assist (MMX, SSE, AVX, GRO, TSO…)
- Public clouds not offering this yet

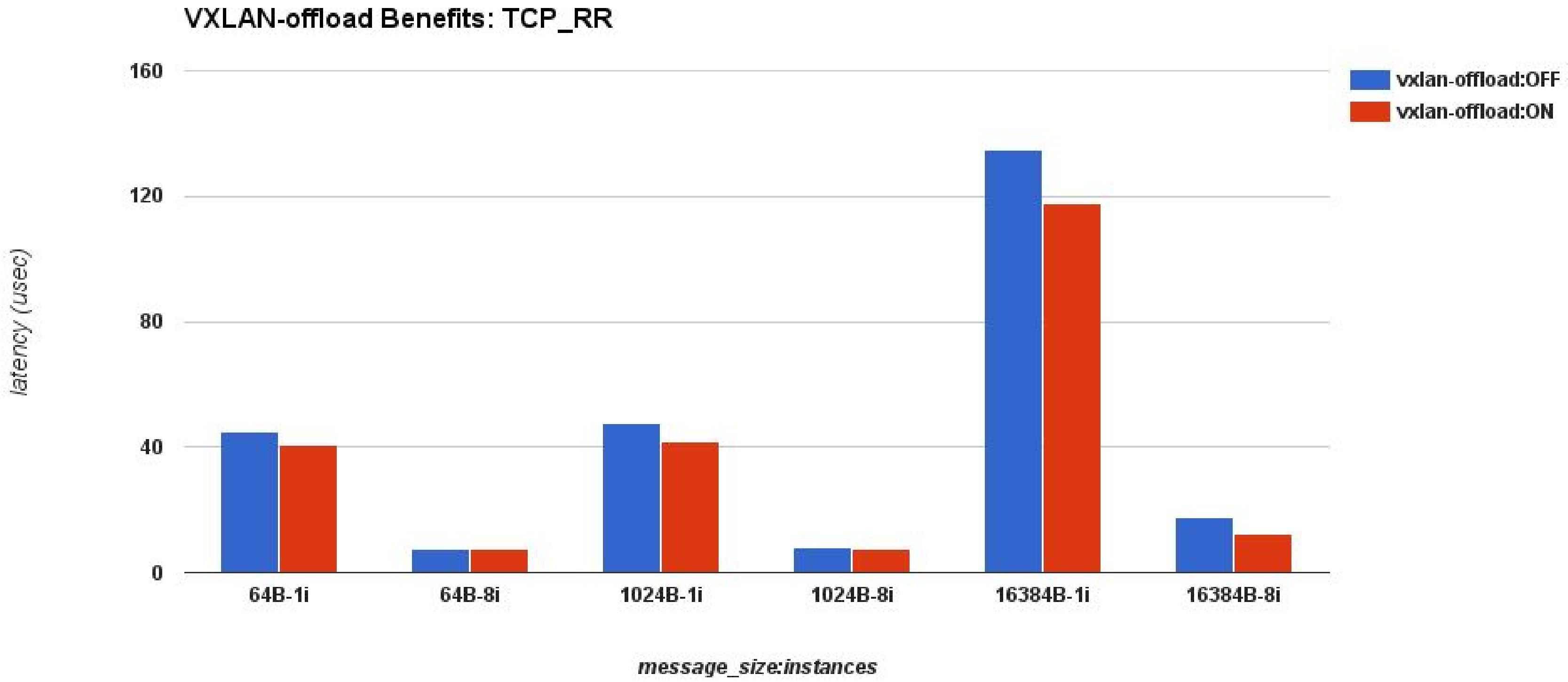
Bare Metal

Benefits of VXLAN-offload (throughput)

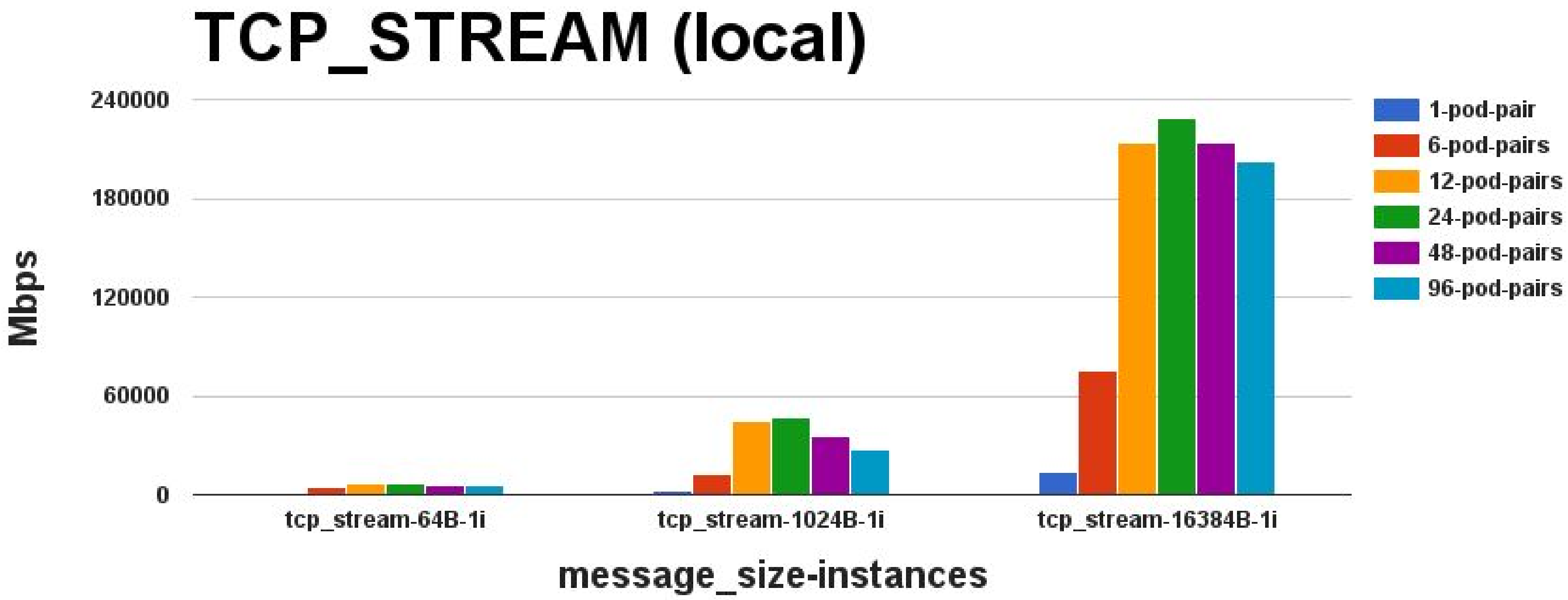


Bare Metal

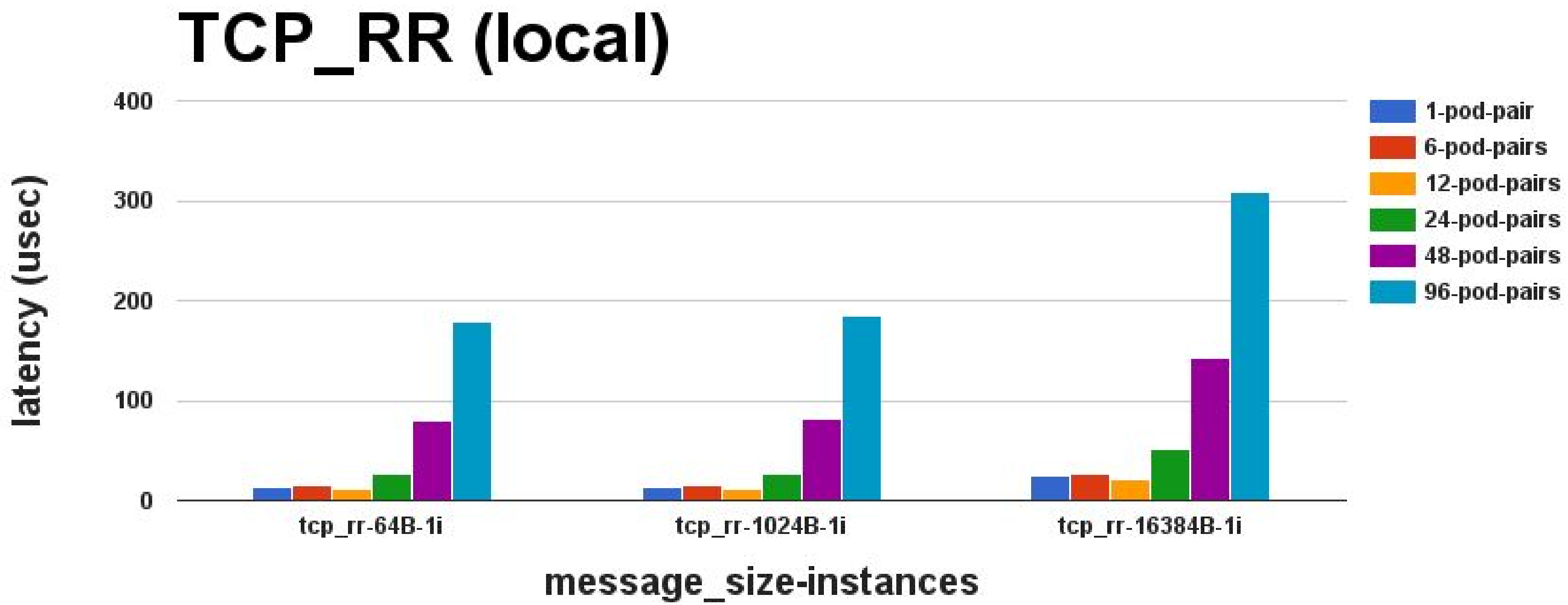
Benefits of VXLAN-offload (latency)



Network Performance (on-box: many pods)



Network Performance (on-box: many pods)



Node Heartbeat Optimization

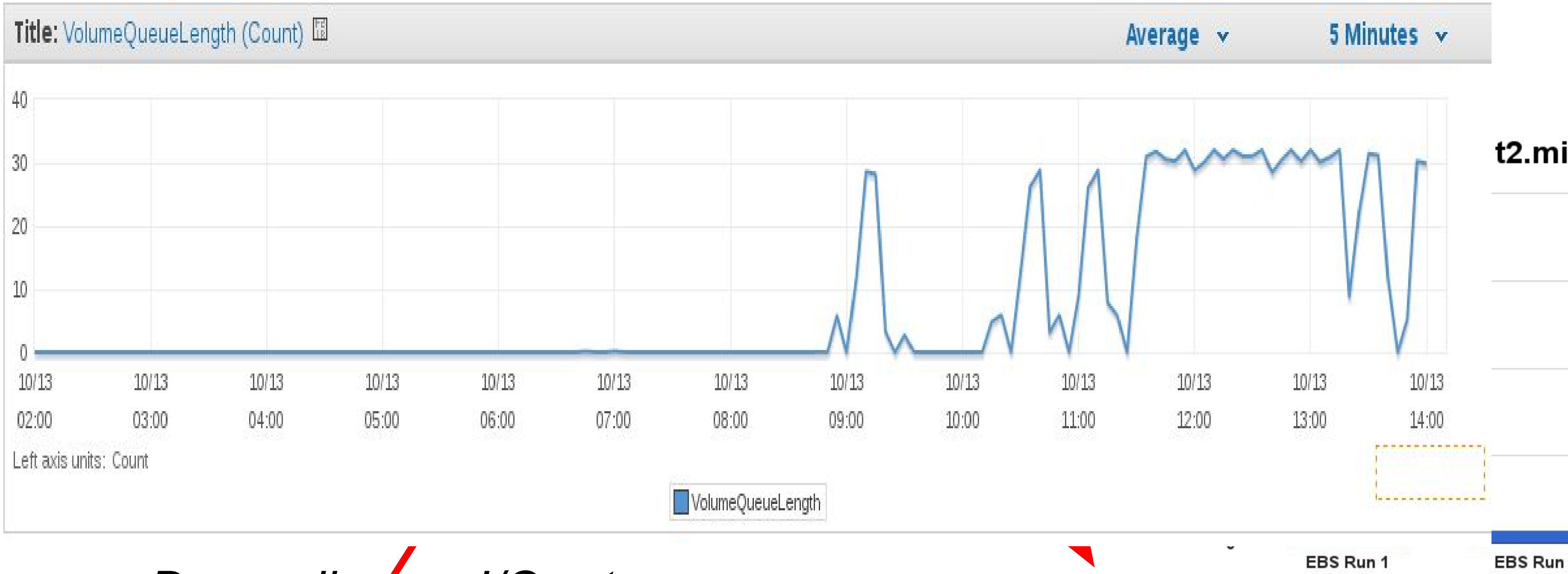
- Cluster network communication shares media with Pod traffic
  - Extreme network load can block cluster heartbeats and lead to node eviction
  - Increase --node-monitor-grace-period in /etc/origin/node/node-config.yml

```
apiServerArguments: null
  controllerArguments:
    node-monitor-grace-period:
      - "120s"
```

Cloud Gotchas...

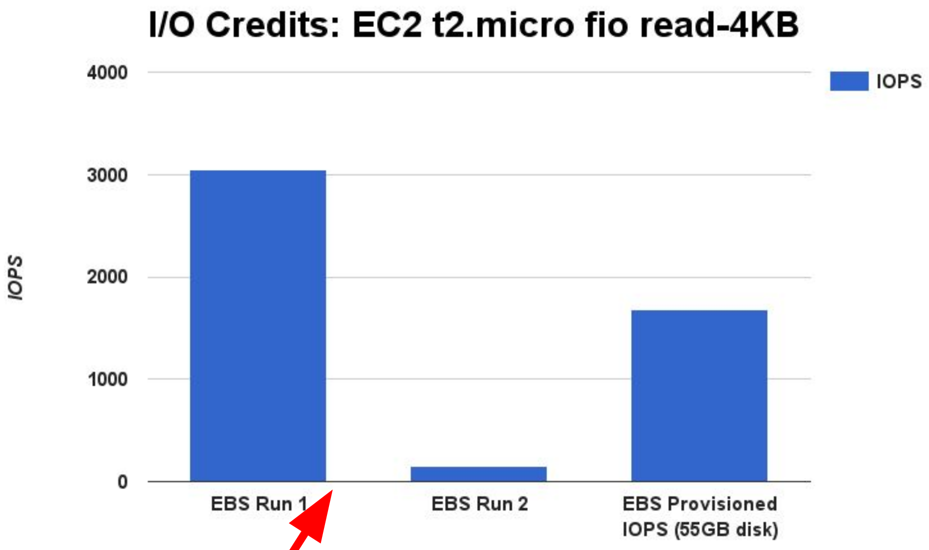
- Variable performance
- Pay-for-performance
- Reset your expectations

Gotcha: EBS I/O Credits/Bursting



Depending on I/O rate, this may never recover

Gotcha: EBS I/O Credits/Bursting



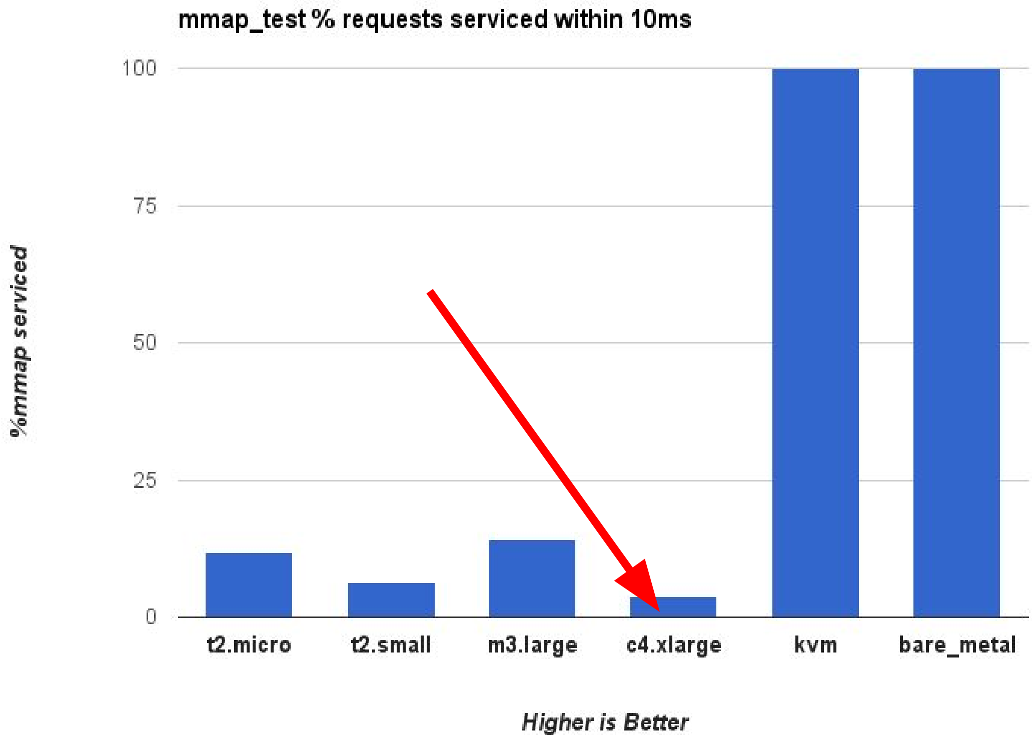
%iowait == 100%
credits ran out

Depending on I/O rate,
this may never recover

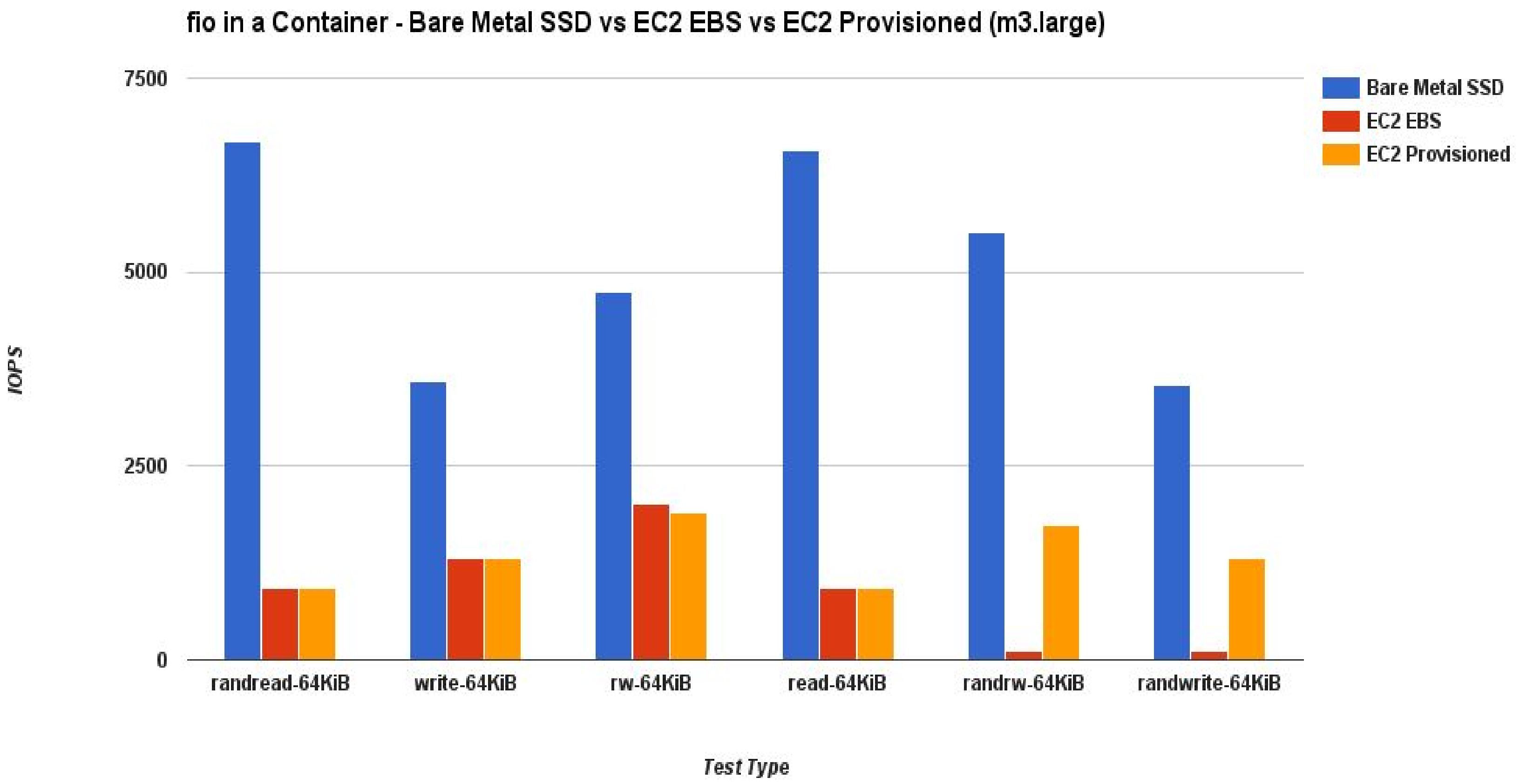
Gotcha: CPU Credit System

Pay for performance?

No...pay for determinism and stability



Storage Performance



Out-of-the-Box Limits

- Kubelet limits to 40 pods per node by default
- OpenShift SDN: 255 nodes, 255 IPs per node
  - Tunable
- Device Mapper backend does not permit page-cache sharing
- Master node does not host pods by default
- Individual “projects” can have Resource Quotas

```
kubeletArguments:
  max-pods:
  - "NN"
```

etcd Tuning

- Needs fast disk (SSD preferred)
- Uses RAM for snapshots...efficiency and performance improvements coming
  - snapshot efficiencies
  - reduction in garbage collection pauses
- https://github.com/coreos/etcd/tree/master/Documentation/benchmarks
- Avoid swap
- Optimize connection between etcd and master
  - Or co-locate them on the same machine

Kubernetes Pod Manifests

- Recipe for how to deploy an application

- Some tuning options are exposed through Kube

- Encapsulate tuning inside script.sh (numactl)

```
pseudo-code manifest:
  - image: my-app:1.0
    securityContext:
      privileged: false
      capabilities:
        add:
           - CAP_SYS_ADMIN
    command:
    - /your/script.sh
    volumeMounts:
    - mountPath: "/perf1"
```

Scheduler Options

/etc/origin/master/scheduler.json

MatchNodeSelector # land a pod on certain nodes

PodFitsResources # ensure sufficient resources to run a pod

PodFitsPorts # ensure ports are available

NoDiskConflict # enforce single writer

Region serviceAffinity labels=region # keep services in same region

LeastRequestedPriority weight: 1 # favor less-committed nodes

ServiceSpreadingPriority weight: 1 # spread pods between nodes

Zone", "weight" : 2, serviceAntiAffinity label=zone # keep service on different zones

Profiling OpenShift (golang)

- Append `OPENSHIFT_PROFILE={cpu,mem,web}` to `/etc/sysconfig/openshift-master`

```
# systemctl restart openshift-master
# systemctl stop openshift-master
# go tool pprof /bin/openshift /var/lib/openshift/cpu.pprof
# top10 -cum
```

- Example: https://github.com/openshift/origin/issues/5106

https://github.com/openshift/origin/blob/master/HACKING.md#performance-debugging

@ Scale
Deployment matters

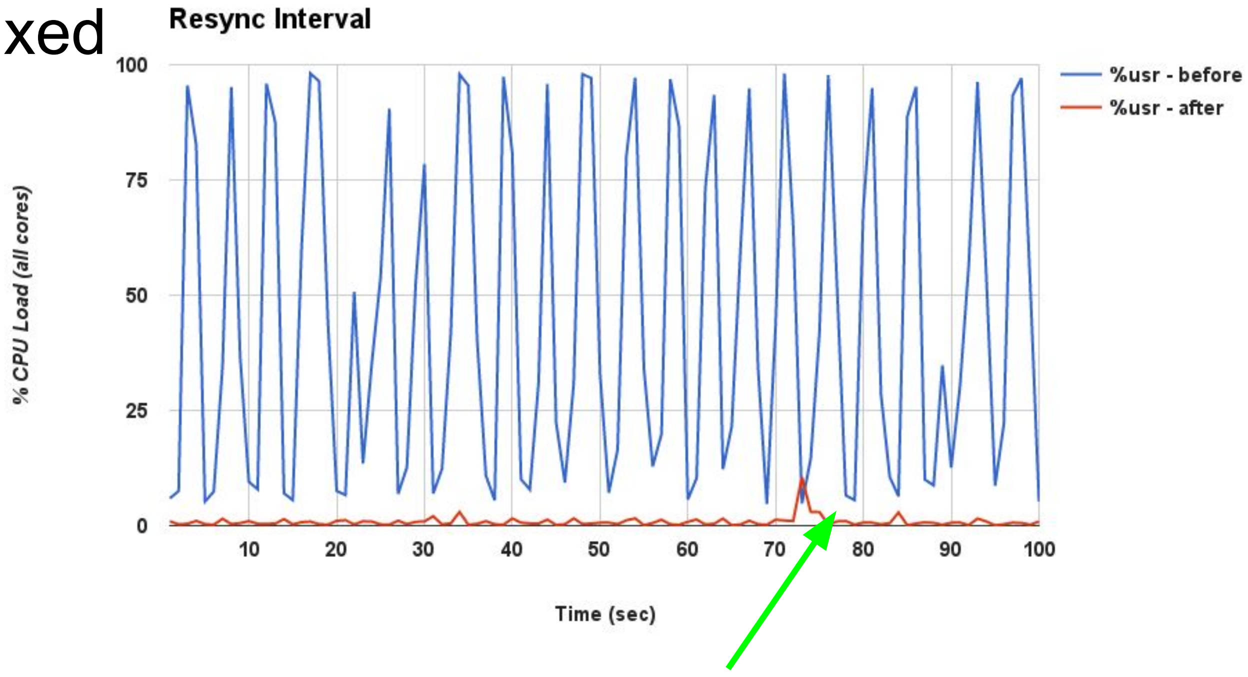
Scaling Up O(10^2+)

HA is important

- master-api & master-controller becomes a single point of failure and has caps to prevent system overload.
    - Load balancing (master-api)
    - active-passive on master-controller
    - in-flight-request limit: 400

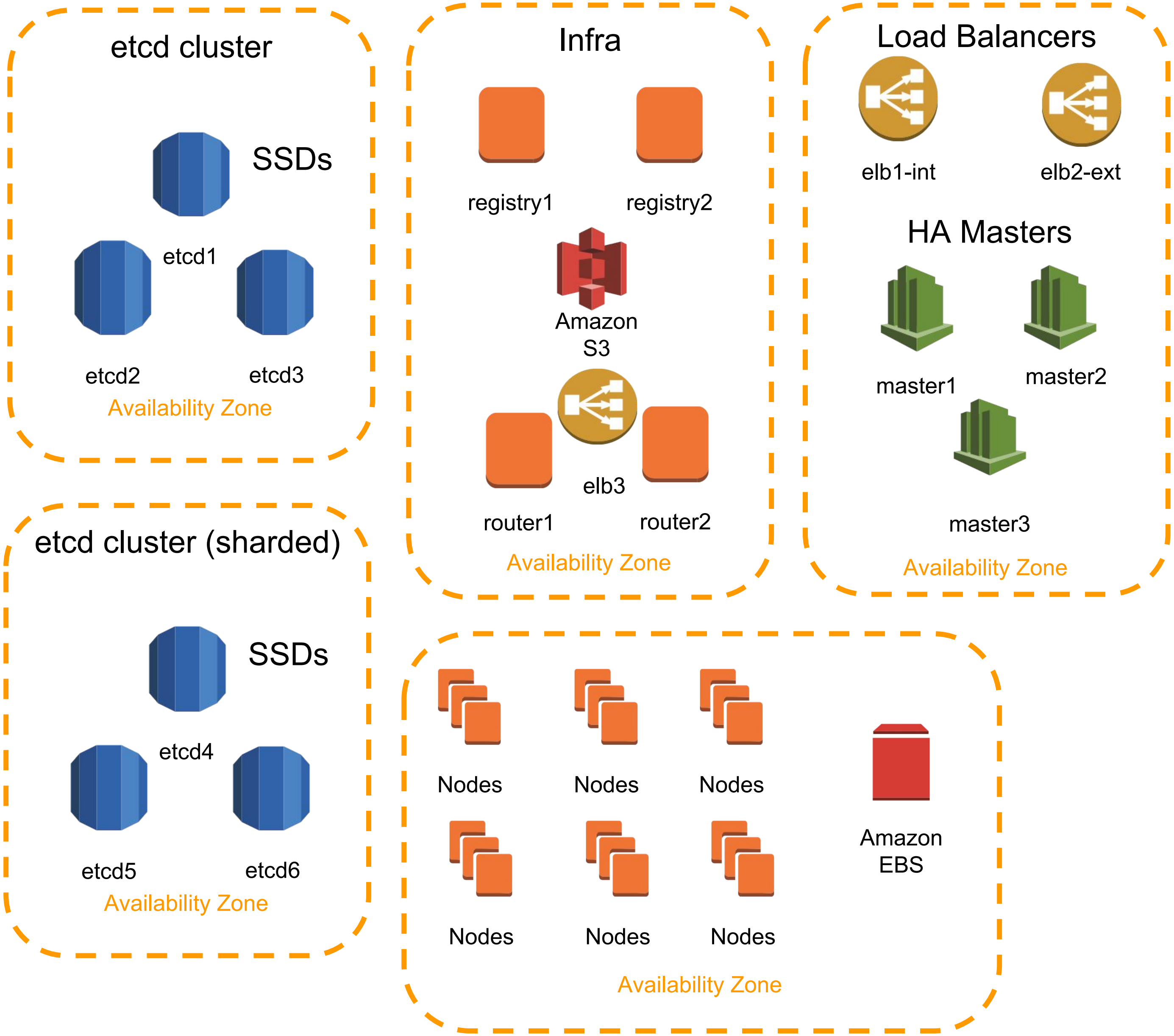
Kubernetes Resync Interval

- Identified and fixed a periodic load spike that limited scale
- Issue #5106



Spike reduced in both frequency and magnitude

OpenShift v3 HA, Scale-out Architecture on EC2



Instance Types

etcd: m4.4xlarge

masters: c4.4xlarge

registry, routers: m4.xlarge

nodes: m4.xlarge

Networks

NET1: Management

NET2: VXLAN - Nodes, Routers, ELBs, Masters

NET3: Dedicated Master←→ etcd link

NET4: Backups?